Linux Audio Conference 2009

3D-Audio with CLAM and Blender's Game Engine

Natanael Olaiz, Pau Arumí, Toni Mateos, David García

BarcelonaMedia research center
Barcelona, Spain

Talk outline

- Motivation and goals
- The big picture
- Blender and its Game Engine
- CLAM and its 3D-audio engine
- The systems communication via SpatDIF
- Game demo (video)
- Conclusions and future work

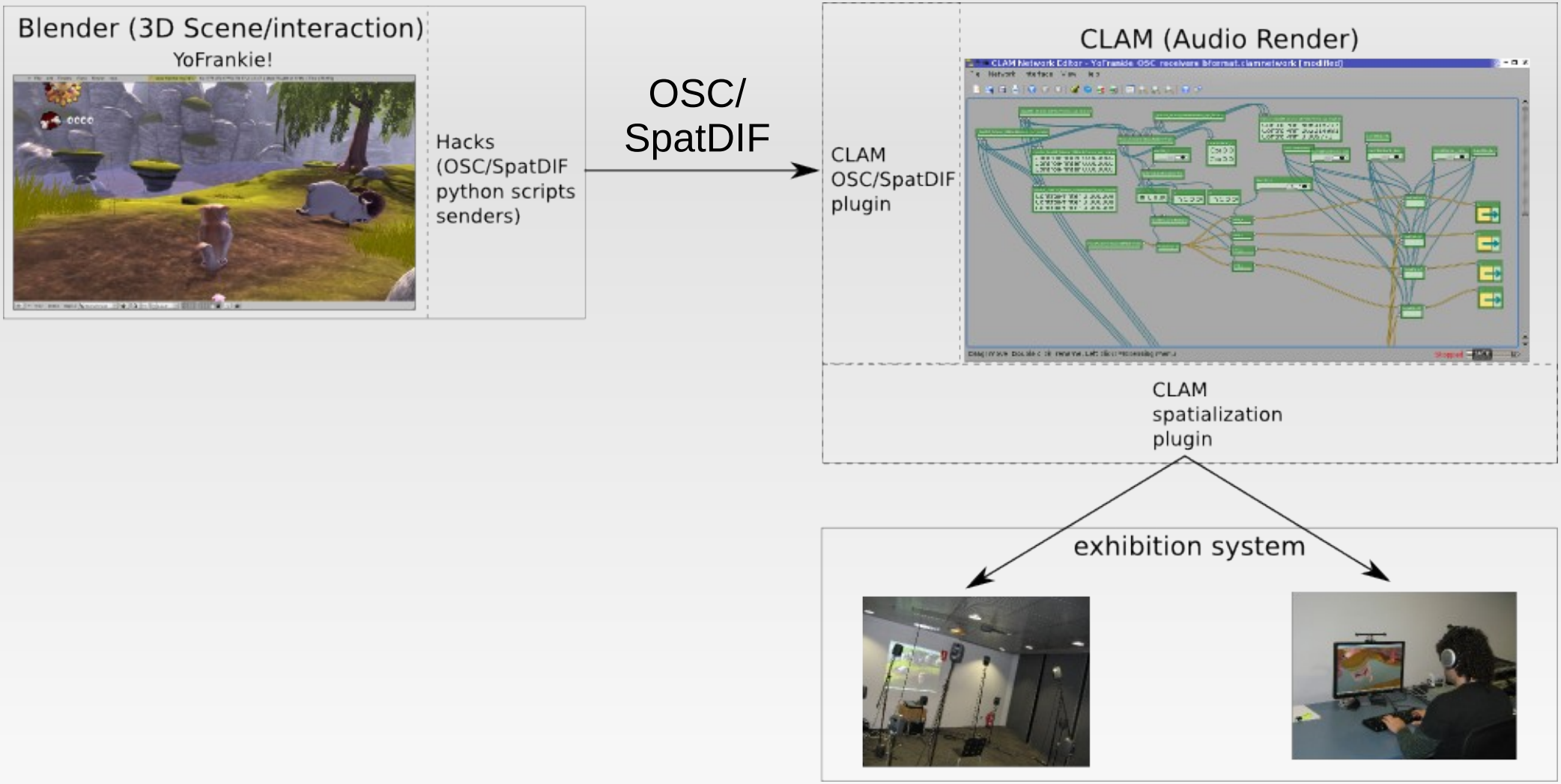
Motivation

- Blender has produced impressive demonstrators:
  - *Elephant Dreams* and *Big Buck Bunny* 3D movies
  - *Yo Frankie!* 3D game
- We have recently developed a 3D-audio platform based on the CLAM framework
  - Room acoustics simulation, ambisonics, binaural, Vector Based Amplitude Panning, etc.
- We faced the need for powerful 3D geometric tools to drive the 3D-audio rendering

Goals

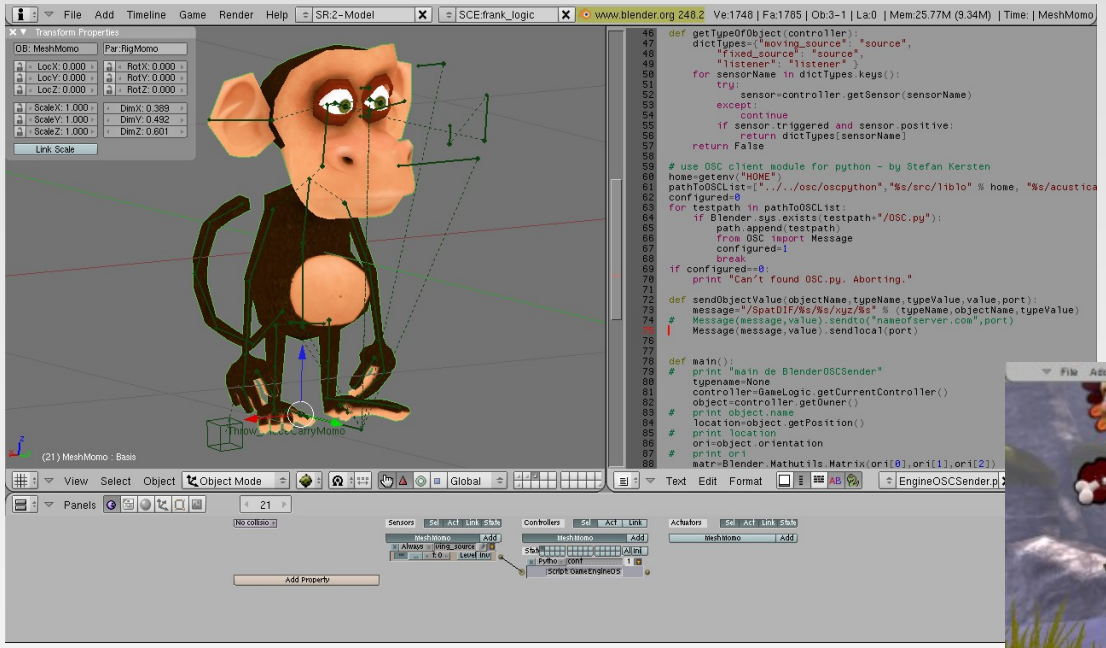
- Take advantadge of Blender!
- Experiment with 3D audio games
- Set up an **experimental platform** to do 3D audio scene rendering
  - Start with two decoupled systems
  - Using OSC to link the geometric scene with the audio rendering → lucky strike: SpatDIF protocol
  - Allow changing the exhibition system and rendering algorithms in CLAM
  - Allow changing the graphics engine and 3D scenes

The big picture



Blender and its Game Engine

Authoring

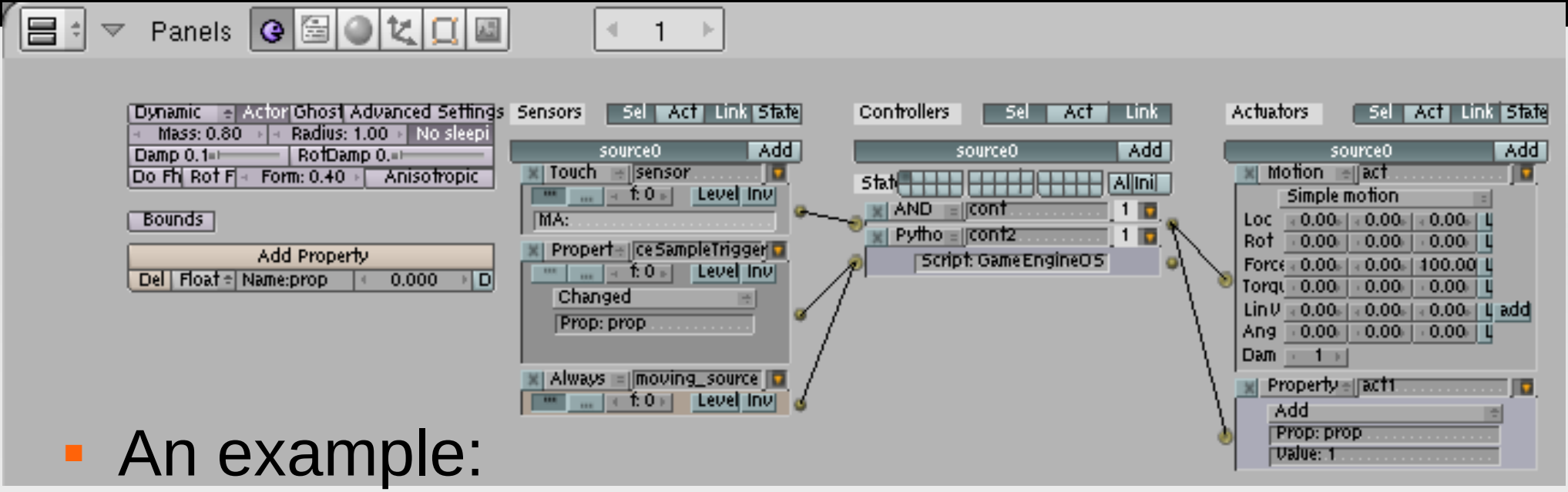


Play

Blender's Game Engine

- Real-time graphics rendering, physics simulations
- Interactivity defined via blocks design
- (Easily) extensible via Python scripting
  - *GameLogic* Python module
- Game logic defined by:
  - *Sensors*: generate events
  - *Controllers*: combine events and trigger actuators or Python scripts
  - *Actuators*: do actions to interact with the scene

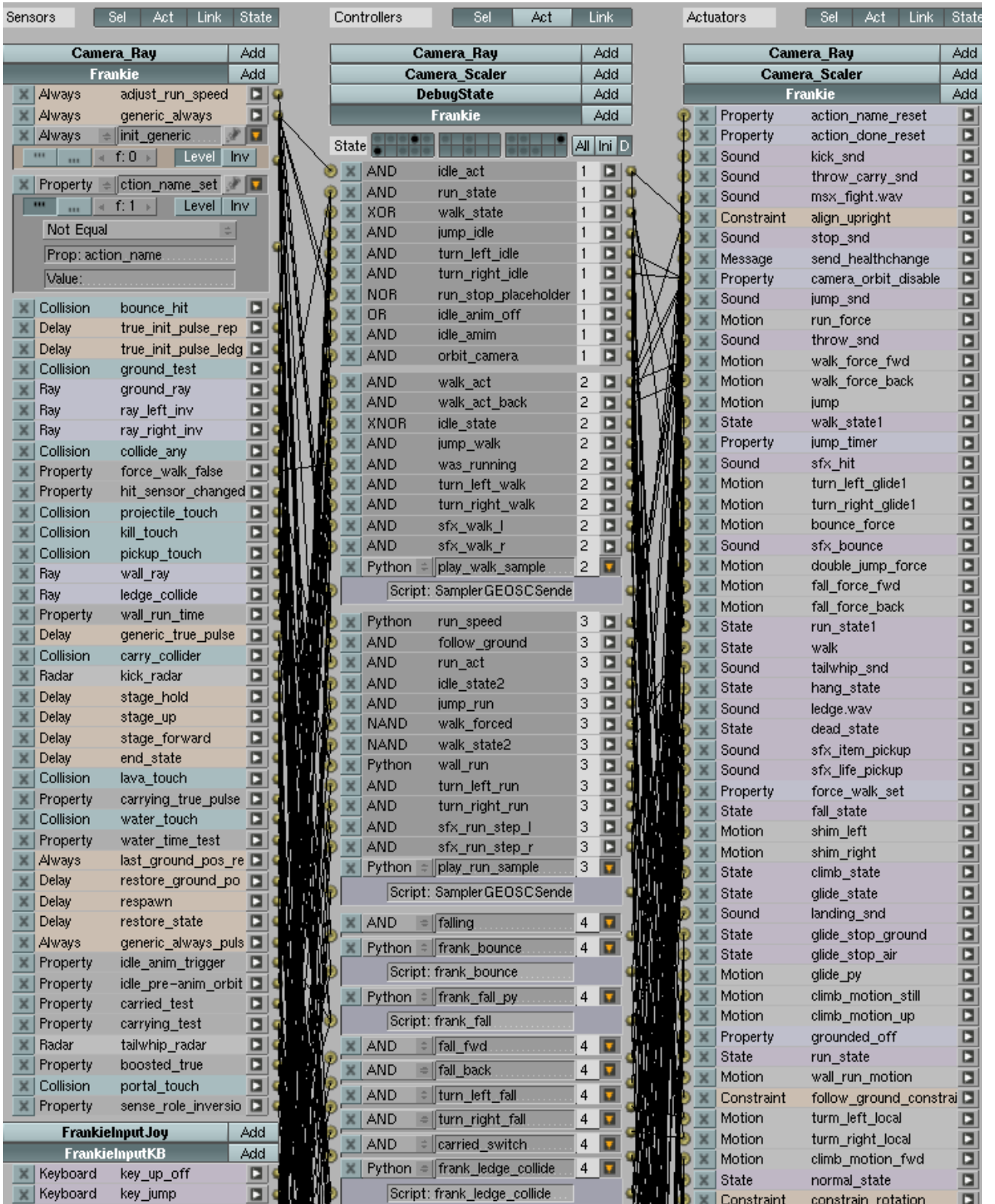
Blender's Game Engine



- An example:
  - A ship object sensor detects a contact with an asteroid object
  - The controller receives the event and, since its state is ”whithout shield” it passes the event to the attached actuators.
  - The actuator receives the event and applies a motion change and decreases an ”energy” prop.

The ”Yo Frankie!” open game

- Yo Frankie! Game logic is defined like this
- And stored in binary format
- This is too entangled to maintain!
- We expect this to improve in the future



Barcelona Media

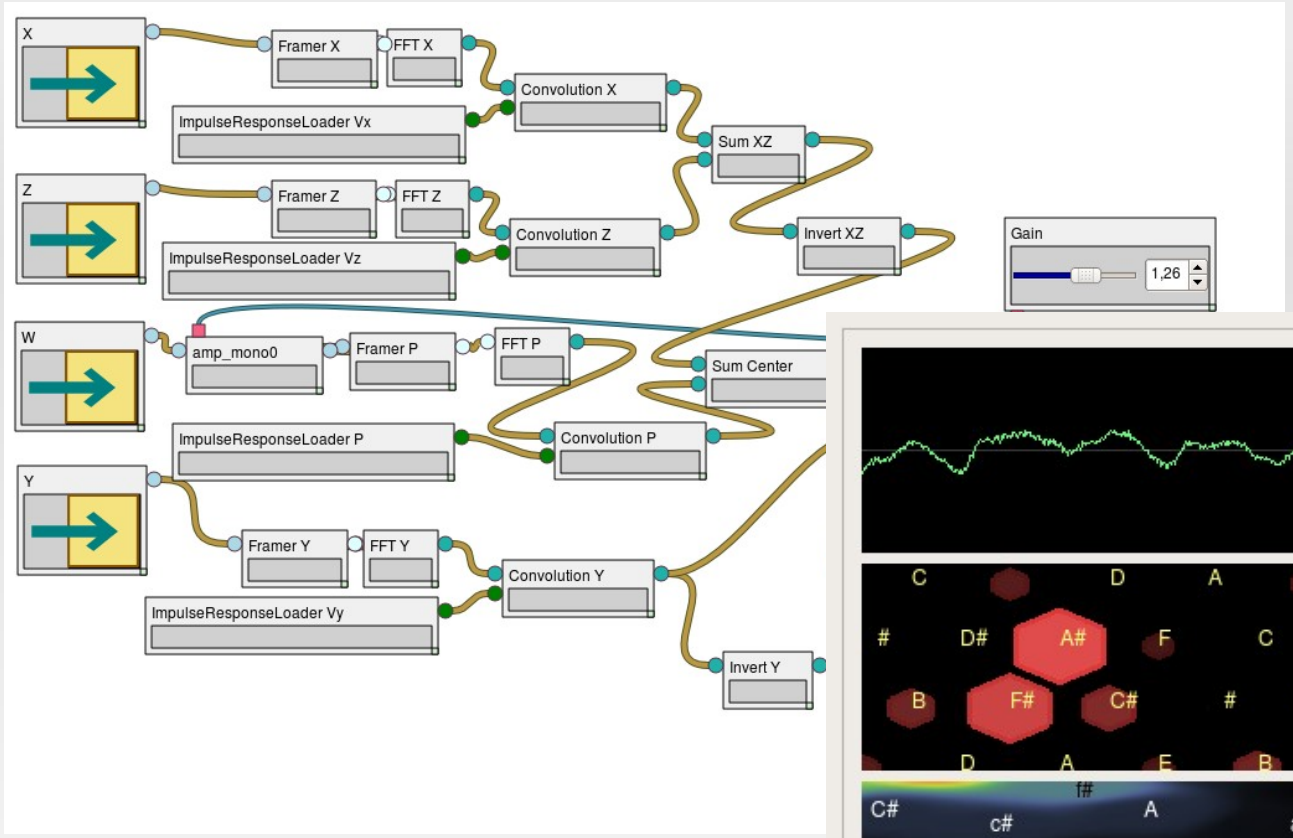
CLAM, the audio framework

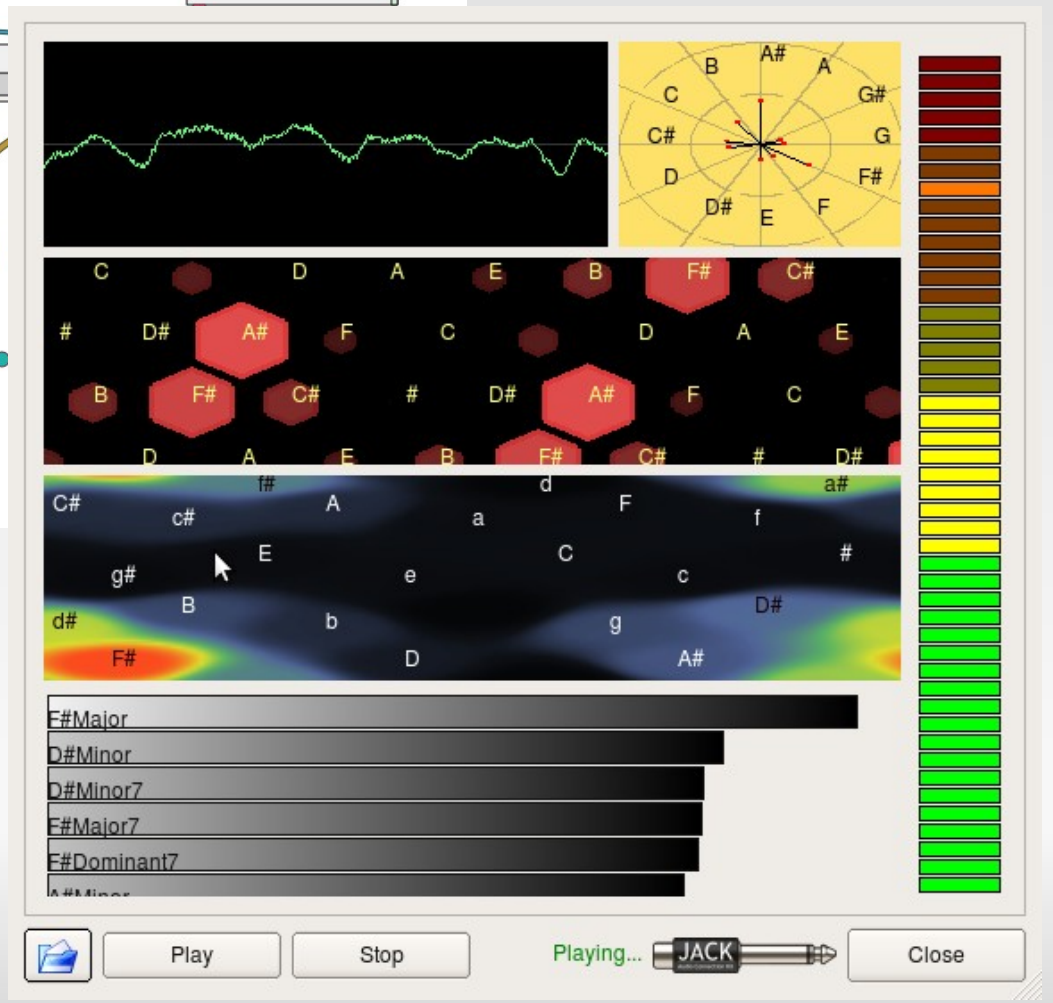
- CLAM stands for C++ Library for Audio and Music
- Started 8 years ago at Universitat Pompeu Fabra / MTG,
- Now a comunity project, mainly used and supported by BarcelonaMedia
- BTW, new URL: http://clam-project.org

Some CLAM features

- Supports different & extensible token types (e.g. Samples, spectrums, features)
- Processing networks are multirate (e.g. processing A runs 3 times for each run of B)
  → Synchronous Dataflow scheduling
- Multiple audio backends: Jack, Portaudio, LADSPA, VST
- Rapid prototyping (via CLAM NetworkEditor and QtDesigner)
- Offline operation (combined with scripting)

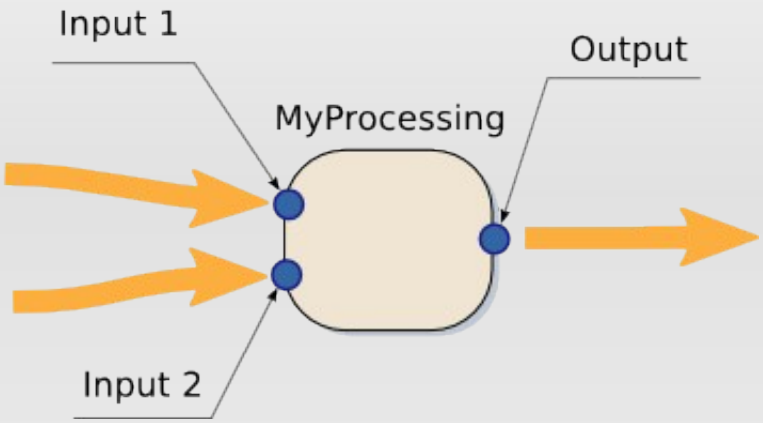
CLAM prototyping

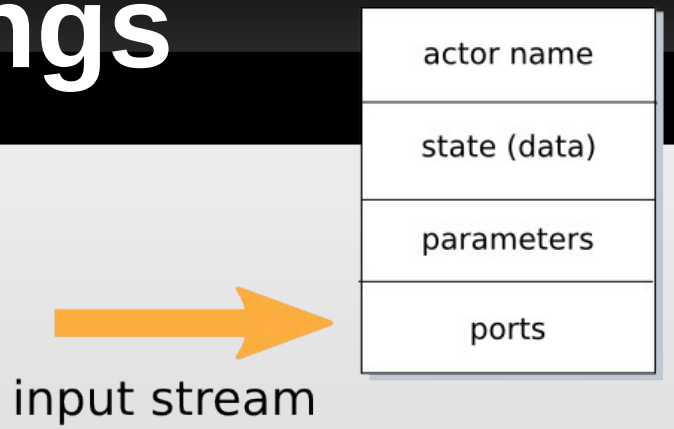


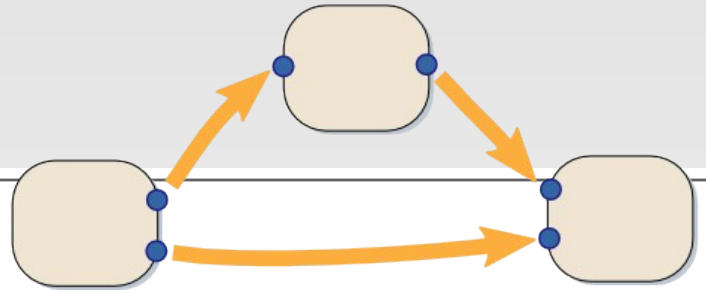


Barcelona Media

CLAM's processings







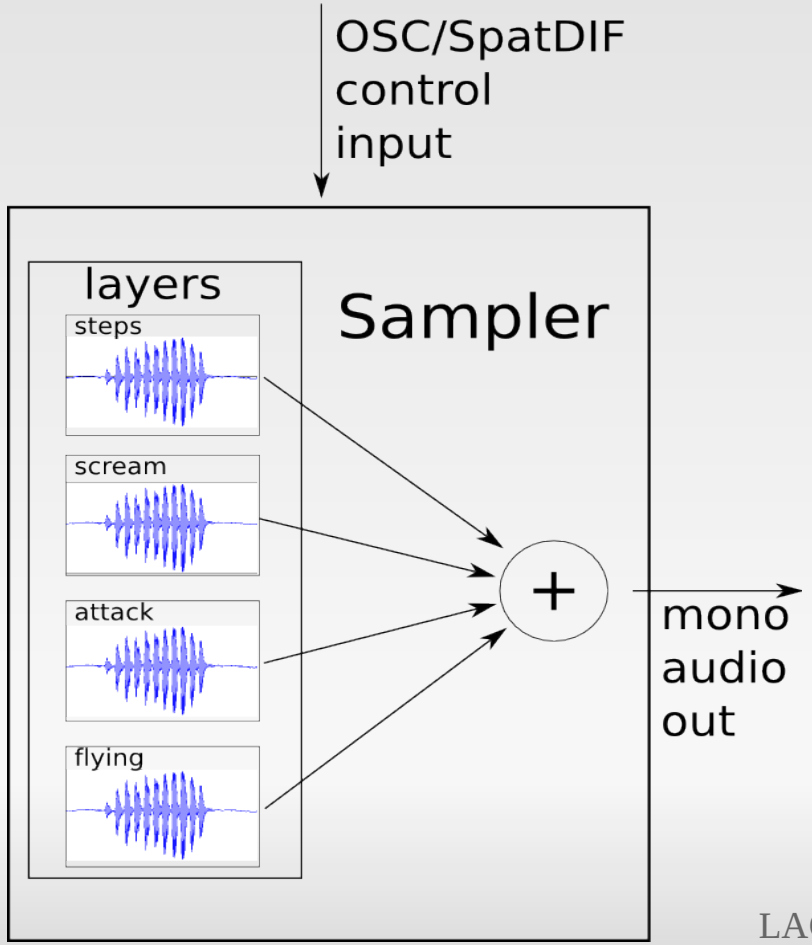
```
class MyProcessing : CLAM::Processing
{
  InPort<TokenType> _input1;
  InPort<TokenType> _input2;
  OutPort<TokenType> _output;

public:
  Adder() : _input1("Input 1",this), _input2("Input 2", this),
            _output("Output", this)
  {
  }
  void Do()
  {
      _output.produce( _input1.consume() + _input2.consume() );
  }
  ...
};
```

CLAM's 3D-audio engine

- **Audio objects** belonging to a scene are **encoded** into a convenient format
  - Audio objects and listener are then animated via SpatDIF messages
  - Audio objects can be **synthesized** via a sampler triggered by SpatDIF messages, or linked to audio streams
- The audio is then **rendered** and **decoded** to the choosen exhibition system
  - Direct-sound and reverb (if exists) are treated differently

Audio objects synthesis

- CLAM processing able to trigger sounds in multiple layers/voices, for each audio object



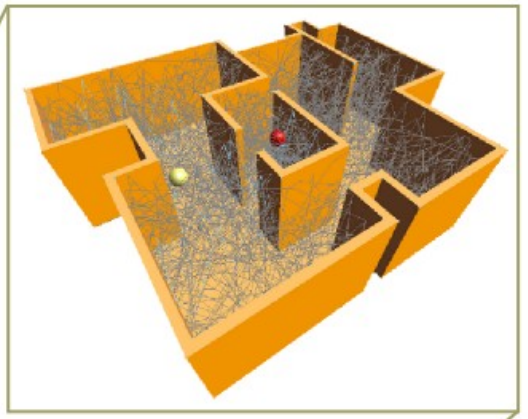
3D-audio rendering & decoding

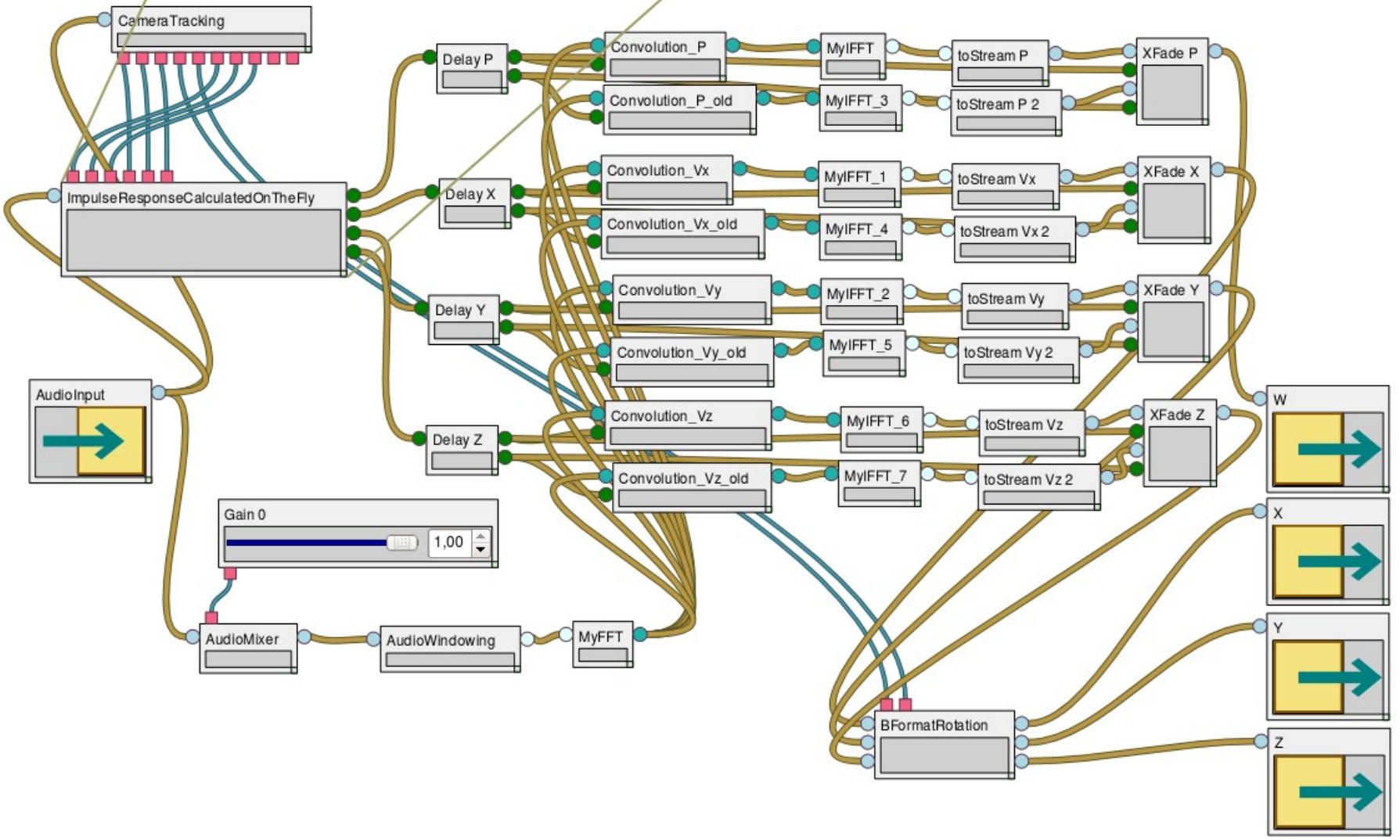
- Direct-Sound
  - Binaural, via a the best-matching HRTF filter
  - (Any) Multi-loudspaker setup, using Vector Base Amplitude Panning (VBAP)
- Reverb (optional)
  - Simulated with a ray-tracing algorithm (developed at BarcelonaMedia but not open-source)
  - But the ”navigation” though impulse-responses is open-source
  - Encoded into Ambisonics (typically 1rst order)
  - Decoded into binaural (using many virtual loudspeakers) and into multi-loudspeakers.

CLAM's 3D-audio engine

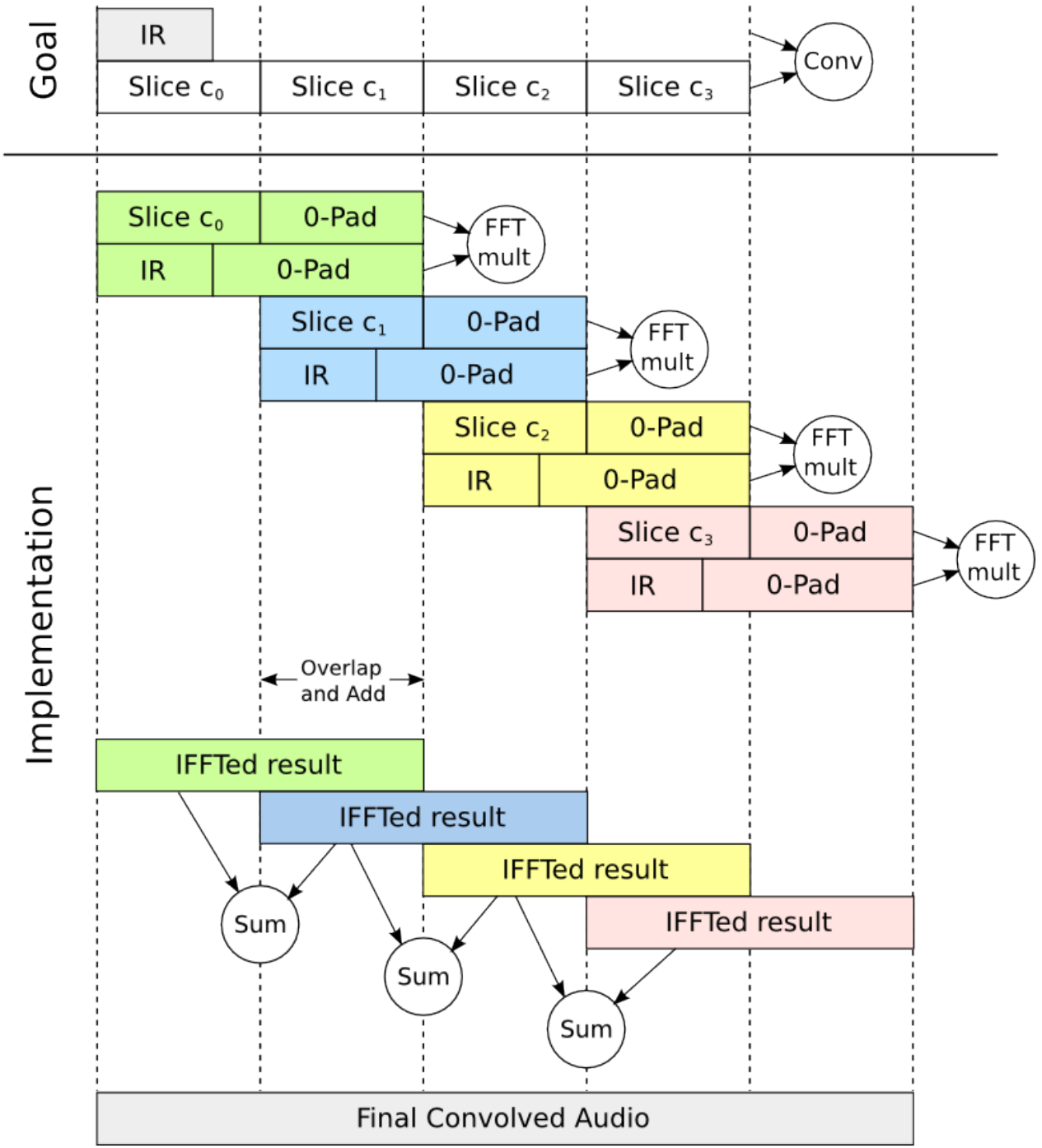
- Demo ray-tracing
- Next slides: rendering and decoding networks
  - Binaural rendering and decoding
  - Bformat (1st order Ambisonics) decoding to surround

- Reverb simulation CLAM network





- Partitioned Convolution



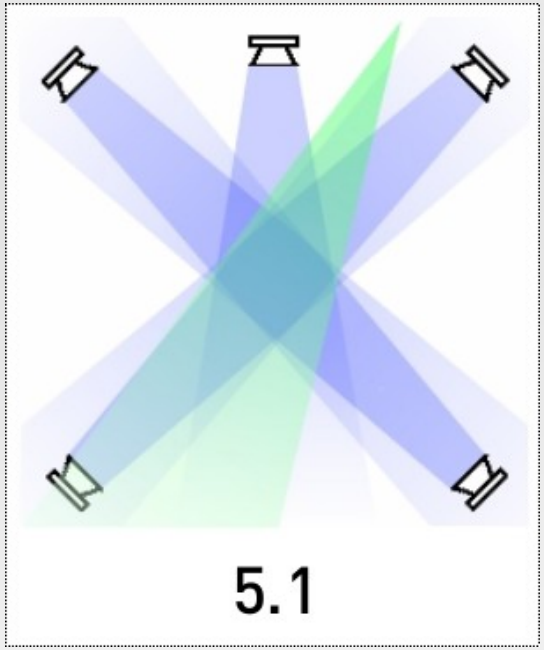
Decoding options

- Binaural (based on HRTFs) with head-tracking

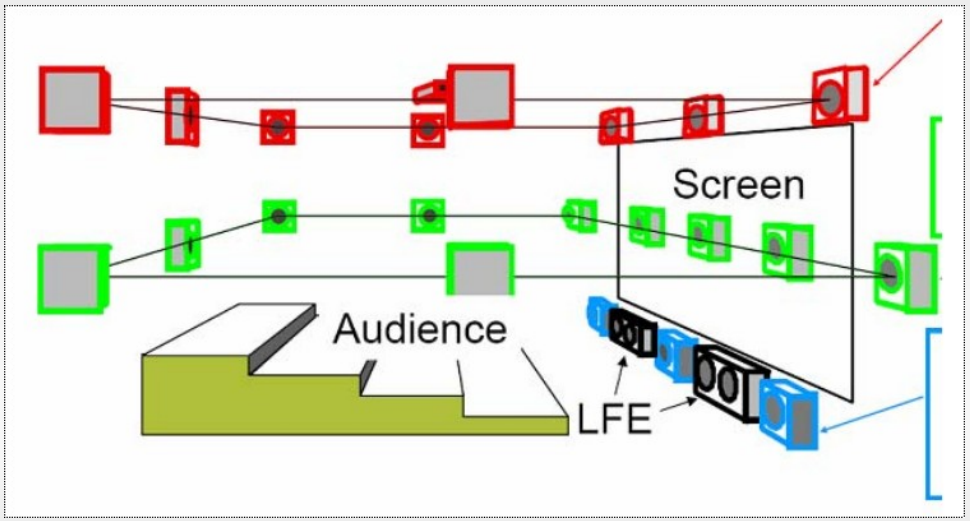
More decoding options

- 15 speakers in our lab (4+6+4+1)

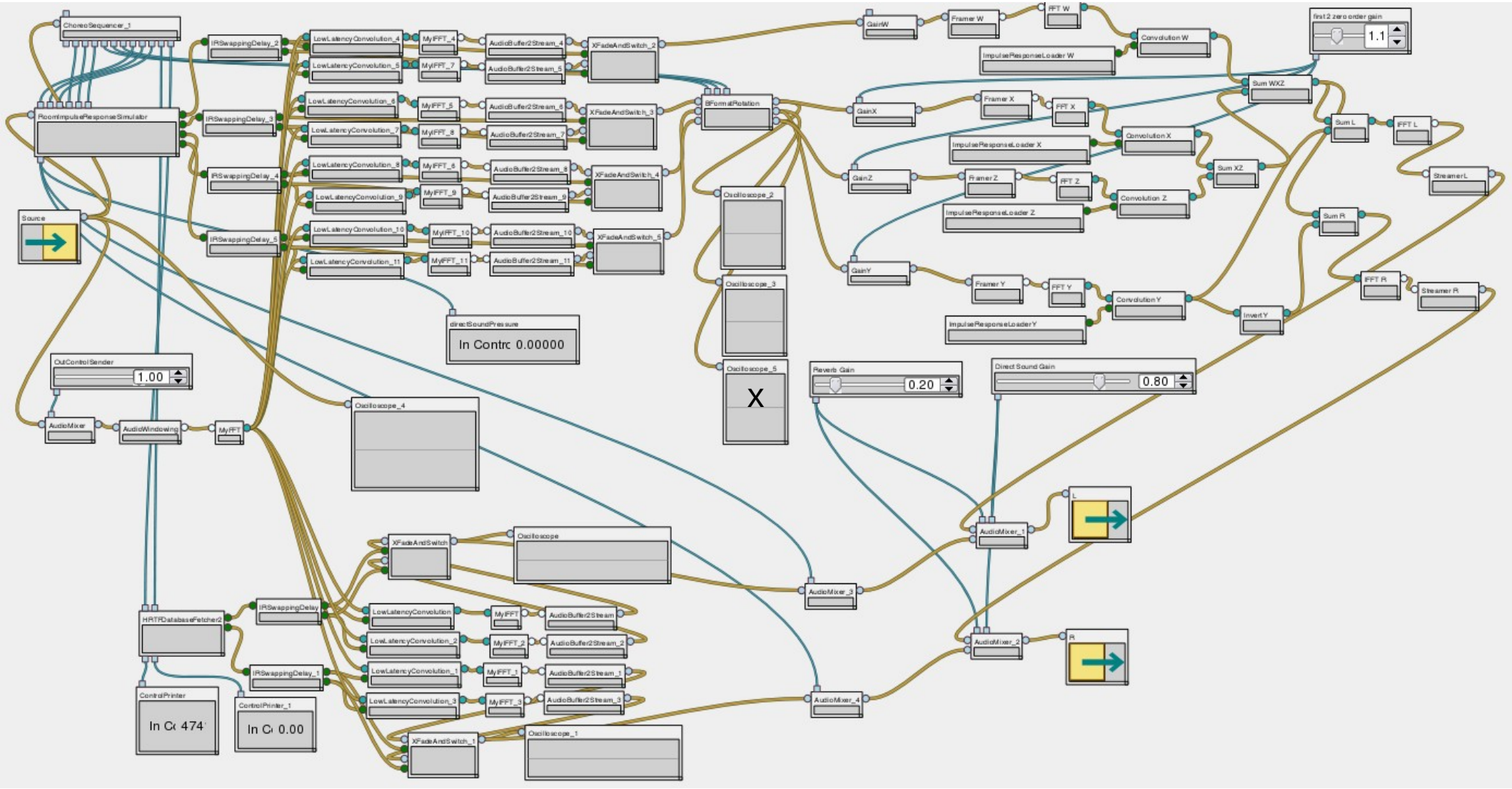
More decoding options



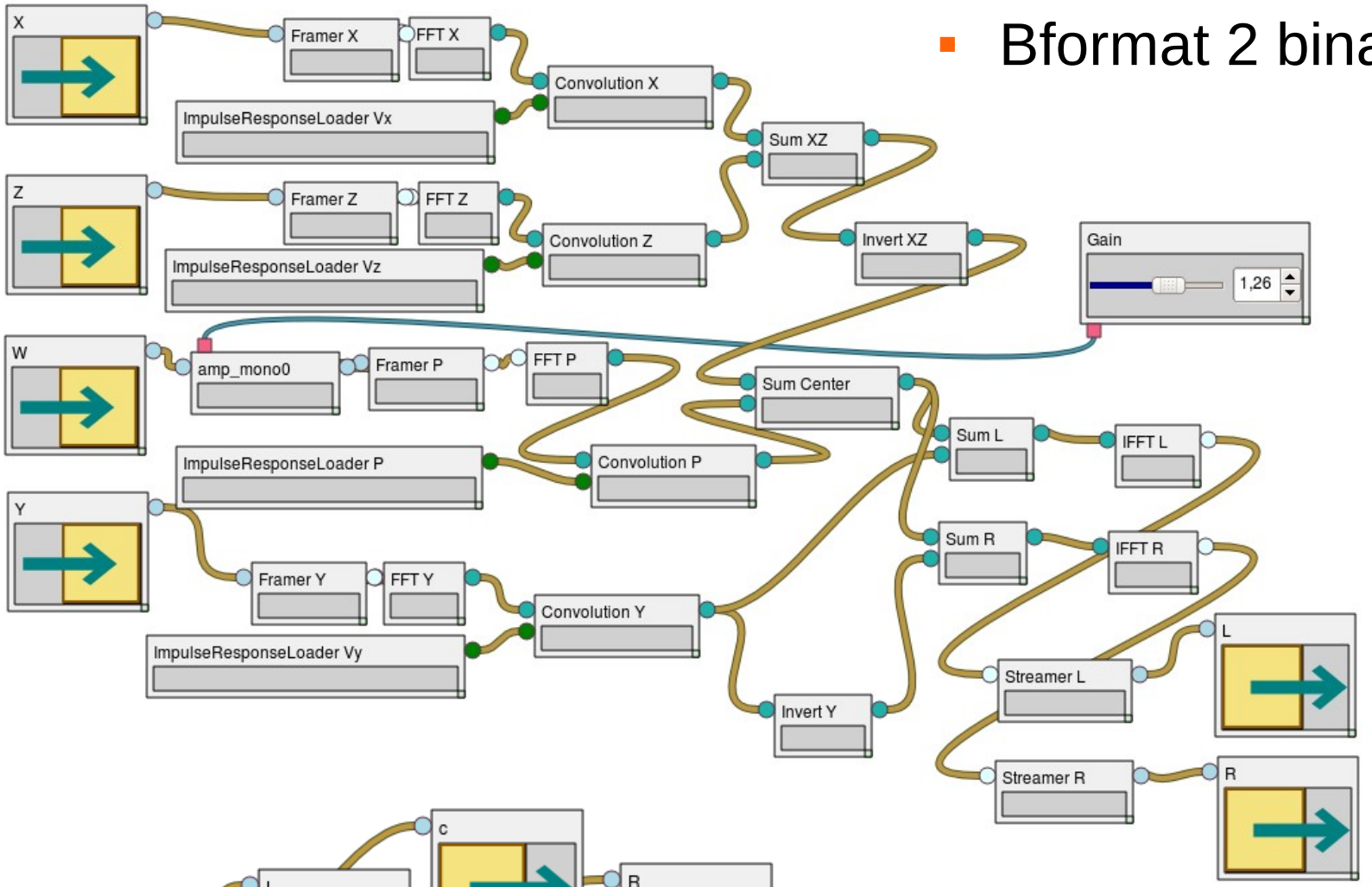
- 5.1, 22.2, etc
- Adding new loudspeakers setups is very **simple**. All it takes is a configuration file with loudspeaker-positions

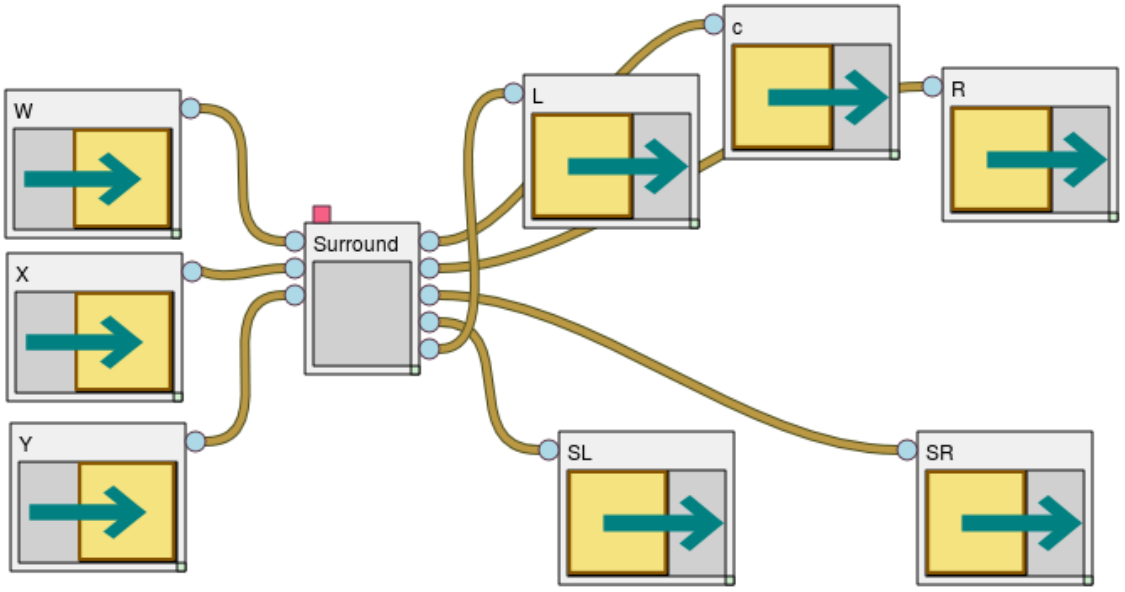


- Direct-sound + reverb simulation in binaural



- Bformat 2 binaural

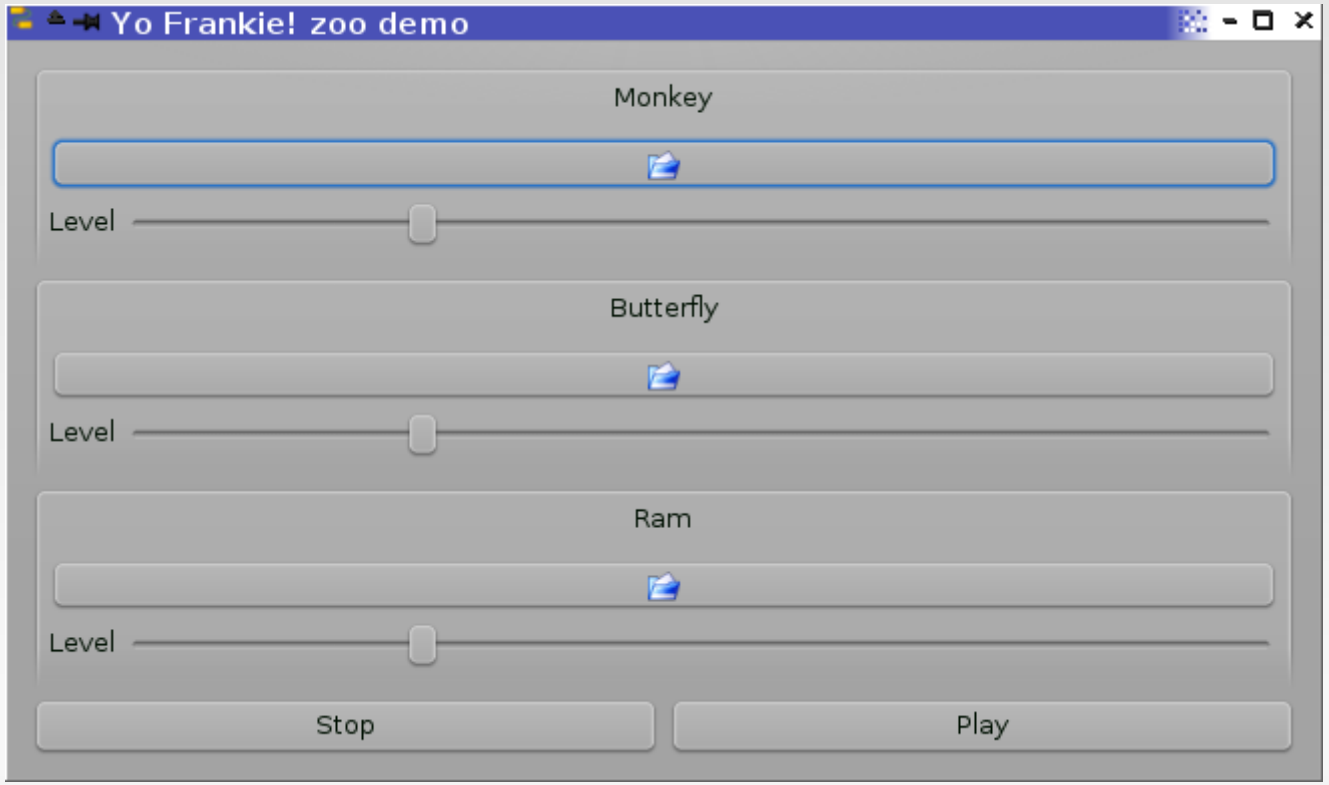




- Bformat to surround 5.0

The audio render user interface

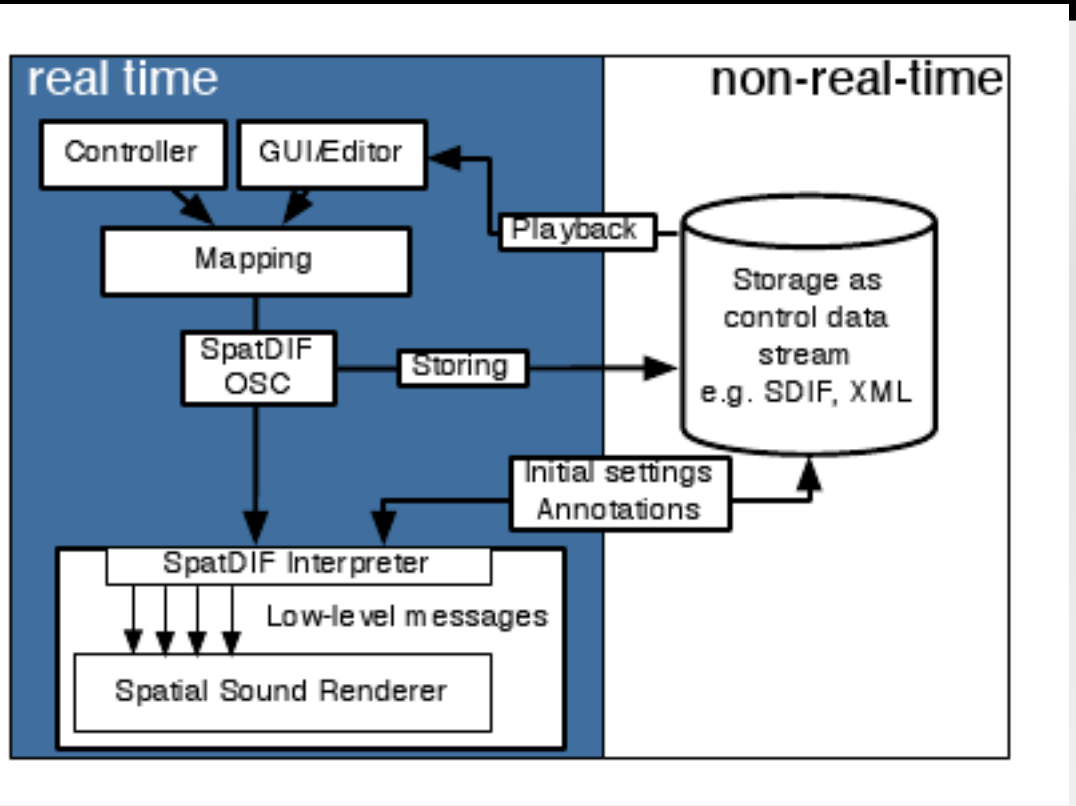
- All the networks are behind the courtains



The SpatDIF protocol

- **Spat**ial Sound **D**escription **I**nterchange **F**ormat
- The definition of the format is work in progress
  - Now it is the right moment to give feedback
- It is built on top of **O**pen **S**ound **C**ontrol (OSC). Which is good!
- The Blender-CLAM system uses a subset of SpatDIF and some **extensions** by our own

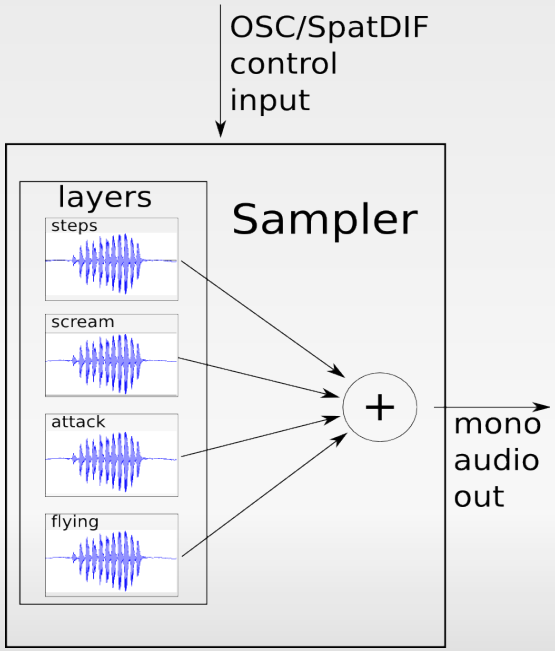
SpatDIF messages examples



- /SpatDIF/source/*n*/xyz (x-axis y-axis y-axis; floats)
- /SpatDIF/source/*n*/aed (azimuth elevation distance; floats)
- /SpatDIF/source/*n*/aer (azimuth elevation roll ; floats)
- /SpatDIF/source/*n*/emissionPattern (cardioid, omni, etc)

SpatDIF extended messages

- /SpatDIF/source/*n*/sampler/addLayer (name ; string)
- /SpatDIF/source/*n*/sampler/*name*/setBuffer (audio file ; string)
- /SpatDIF/source/*n*/sampler/*name*/setLoop (bool)
- /SpatDIF/source/*n*/sampler/*name*/play (start|stop ; bool)



Video demo (in binaural)

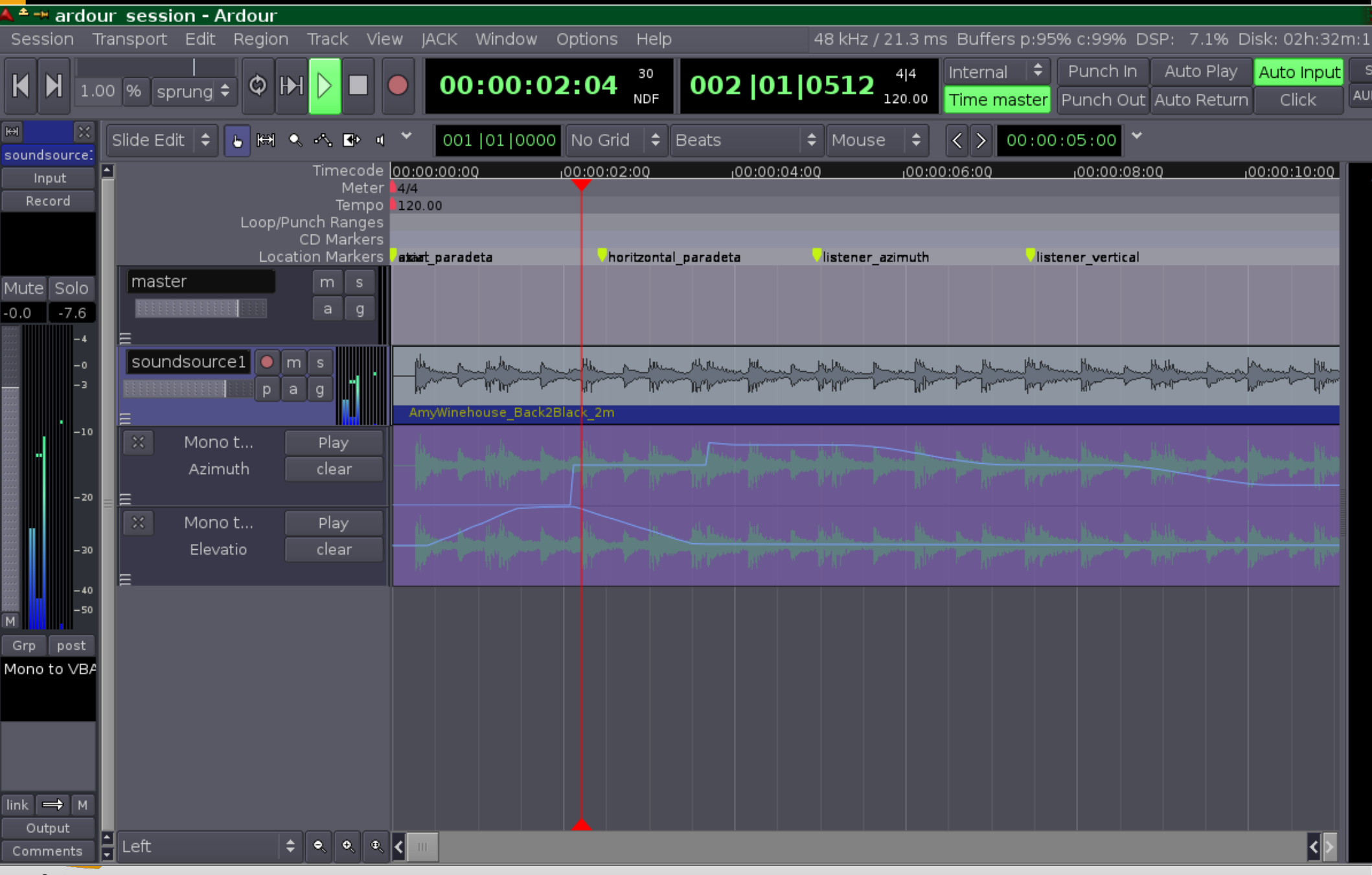
Conclusions

- This work is fun (and we get paid for it!)
- We are learning a lot along the way. Recent improvements:
  - Segregation of direct sound and reverb (VBAP and Ambisonics)
  - Binaural with head-tracking makes a huge difference
  - In postproduction, mixing 10 tracks of 15-channels (VBAP) files in Ardour does not work (HD not so fast)
    - → Use VBAP decoding plugins with automations instead

Conclusions: other given uses

- Beyond Blender's Game Engine
  - Real-time scene manipulation: Link CLAM's audio scene rendering with Blender (not GE), to hear the scene being manipulated
    - Using motion trackers
  - High quality (ray-tracing) offline rendering, using CLAM's offline network player and Python to stablish parameters
  - Export Blender's scene into an Ardour's session (with spacialization plugins with animated controls)
    - Several 3D audio movies segments have been produced

Conclusions: other given uses



Future Work

- Improved scene/samplers setup via SpatDIF
- Better Ambisonics decoder for non-regular setups, shelf filtering and higher order
- Non-punctual audio objects (e.g. a river)
- Encapsulate 3D audio functionality in a well defined library (ala openAL)
- Long term: incorporate geometric metaphors into DAWs
  - Simplified, customized set of Blender's features
  - Production independent of the exhibition system

Thanks!

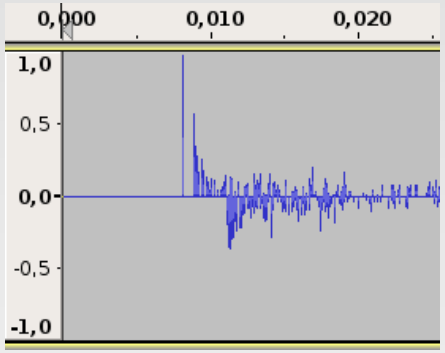
Questions?

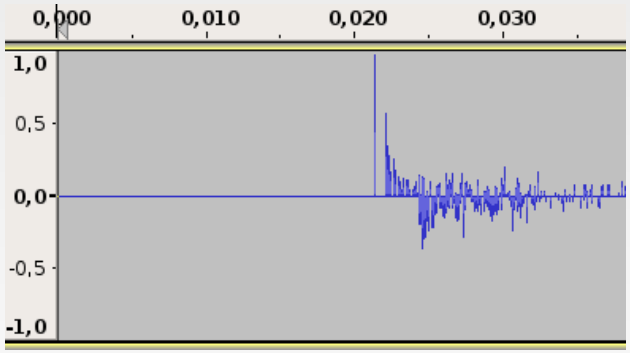
(You are welcomed to the CLAM workshop.
Tomorrow at 15h)

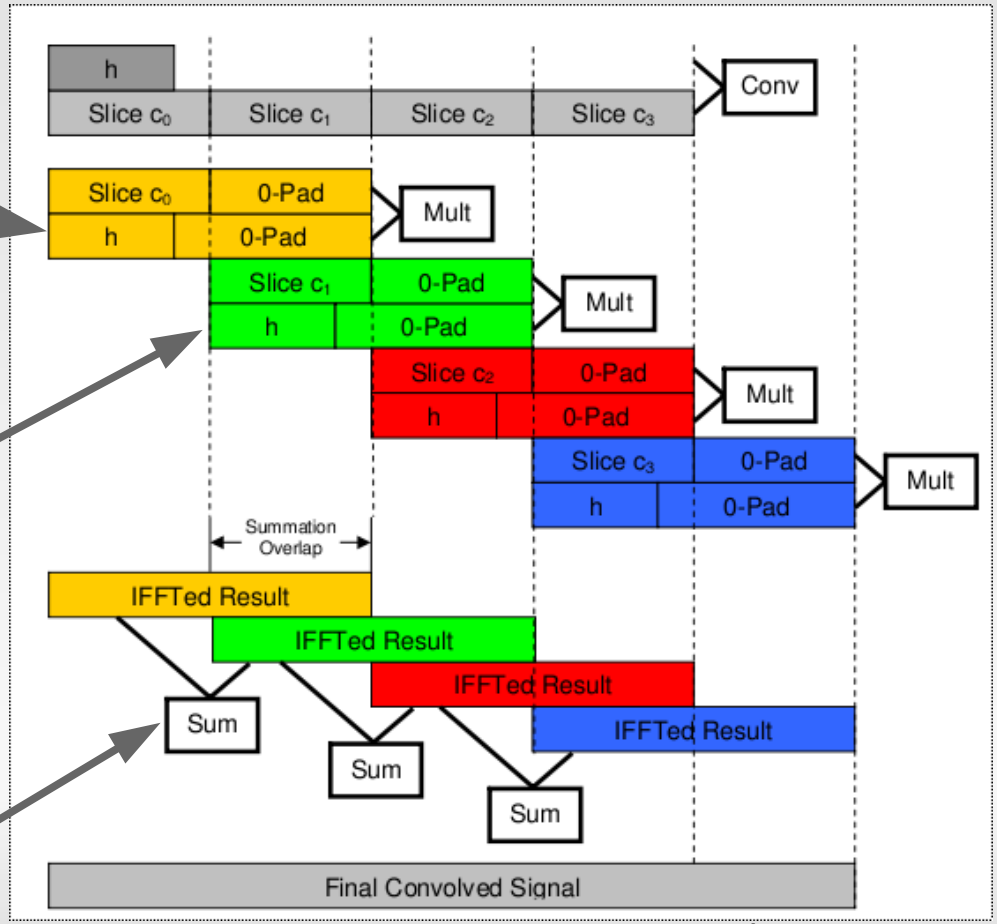
Following slides not used

The devil is in the details

- Switching IR's + IFFT overlapp and add







BIG PROBLEM!! (click)